Distributed delivery system for time-shifted streaming systems

Yaning Liu, Gwendal Simon

Distributed Delivery System for Time-shifted Streaming Systems

Yaning Liu[1,2]
[1]State Key Lab. of Networking and Switching Tech.
Beijing Univ. of Posts and Telecommunications, China

Gwendal Simon[2]
[2]Computer Science Department
Institut Télécom - Télécom Bretagne, France

***Abstract*—In live streaming systems (IPTV, life-stream services, etc.), an attractive feature consists in allowing users to access past portions of the stream. This is called a time-shifted streaming system. We address in this paper the design of a large-scale delivery system for a time-shifted streaming application. We highlight the challenging characteristics of time-shifted applications that prevent known delivery systems to be used. Then, we describe the turntable structure, the first structure that has been specifically designed to cope with the properties of time-shifted systems. A set of preliminary simulations confirm the interest for this structure.**

I. Introduction

The delivery of television over the Internet (IPTV) is expected to offer viewers new ways to enjoy TV content. One of the most promising services, often called *catch-up TV* or *time-shifted TV*, consists in allowing viewers to watch their favorite broadcast TV programs within an expanded time window. Let's say that a program is normally broadcasted from a given time t. In a catch-up TV, this program is made available for viewing at any time from t to $t + \delta$ hours where δ can be excessively long (several weeks). In this context, a viewer is also able to surf the TV content history using pause, rewind or fast forward commands, hence he/she can switch from a live experience to a *shifted* one.

Today, to enjoy catch-up TV requires to record the stream on a *Digital Video Recorder* (DVR) connected to Internet. Of course, this is unacceptable for TV providers, which would like to control the delivery of their content. However, building a large-scale time-shifted streaming service is not trivial. Indeed, the disk-based servers that are currently used in on-demand video services (VoD) have not been designed for concurrent read and write operations. In particular, a VoD server can not massively ingest content. Moreover, delivery systems for IPTV can not be utilized because, contrarily to live streaming systems, time-shifted systems can not directly use group communication techniques like multicast protocols, for the reason that clients require *distinct* portions of the stream. Other obvious differences include the length of a catch-up TV stream, which can be several orders of magnitude longer than a typical movie in VoD, and the dynamicity of chunk request. Contrarily to VoD, the popularity of every chunk is variable in catch-up TV.

A few papers have recently addressed the VCR problem in peer-to-peer VoD systems [1, 2], but no previous work has assumed that VCR is so massively employed by user. Several works have highlighted the problems met by classic centralized architectures [3, 4]. New server implementations are described in [5]. Cache replication and placement schemes are extensively studied by the authors of [3]. When several clients share the same optical Internet access, a patching technique described in [4] is used to handle several concurrent requests, so that the server requirement is reduced. Some works have recently sketched a *peer-to-peer* architecture for time-shifted TV systems. In [6], every client stores all downloaded video parts. A Distributed Hash Table (DHT) is used to keep trace of the owner of every video part, so that a peer that is able to upload a past video part can be found upon a simple request to the DHT. Similarly, a DHT is used to locate video parts in [7]. However, these works appear to suffer from critical drawbacks. First, the use of the hash function seems irrelevant in this context where chunks are iteratively produced. A structure that takes into account the stream linearity would be more appropriate. Second, a peer departure should conduct to multiple deletions in the DHT. For peers that store vast amounts of chunks in catch-up TV, a huge number of messages should be generated. Furthermore, the DHT could not guarantee the availability of all chunks, particularly for early and unpopular chunks.

In this paper, we present a *distributed system* for a time-shifted streaming service. This system tackles several crucial challenges of time-shifted systems. In particular, we ensure the storage of *every* past chunks, we guarantee a good quality of service with a large majority of requests that are fulfilled (past chunks are served to clients), and we balance the load of storing and delivering the chunks to all the peers. Moreover, the fact that the system is fully distributed makes that the system is scalable (an unlimited number of peers can participate) and dependable (no failure point).

Our system relies on a structure, namely *turntable*, which is a lightweight overlay network. We describe the foundations of this structure in Section II. Then, we present in Section III a set of simulations where the quality of this structure is highlighted. In this simulations, we have based on a recent series of measurements in order to build realistic settings. In particular, the behavior of clients has been precisely simulated.

II. The Turntable Structure

We propose a structured system based on a *turntable*. See Table I for summarized notation, and Figure 1 for a represen-

Notation	*Description*
C, ch_i	Set of chunks and ith chunk produced
S, s_i, m	Set of turntable sectors, ith sector, and number of sectors
V, n	Set of peers, number of peers
$\hat{x}_i$	Responsible peer in sector s_i
$\Gamma^{intra}(x)$	Intra-sector neighbors of the peer x
$\Gamma^{inter}(x)$	Inter-sector neighbors of the peer x
c_x	Upload capacity of x
$\Upsilon(z)$	Set of peers serving client z
k	Number of hops for flooding chunk requests
f_j	Number of flooding failures for ch_j
p_j	Probability to forward the fresh chunk ch_j

TABLE I
SUMMARY OF KEY NOTATIONS

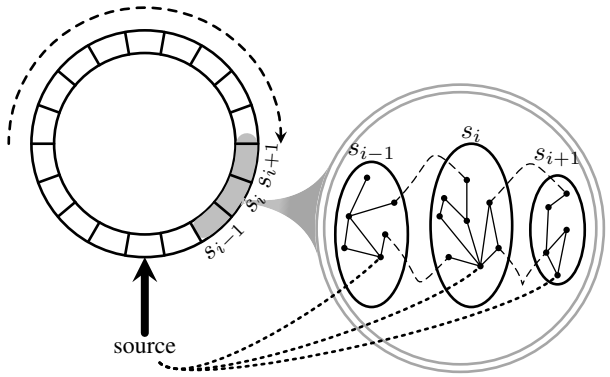

Fig. 1. The distributed turntable structure

tation. The model and protocol are described as follows.

A. Turntable Model

We divide the turntable into m *sectors* noted $s_i, 0 \leq i < m$. Every peer joins exactly one sector. The turntable implements a rotational motion in clockwise direction. At every cycle t, the source produces a new chunk that is sent to a sector s_i, then the chunk produced at cycle $t+1$ is sent to the sector s_j with $j = (i+1) \mod m$, and so on[1]. Hence, every chunk is under the responsibility of a sector, *i.e.*, of a subset of peers. They store the chunks, and deliver them to clients. Please note that a cycle is a generic duration, but a chunk is typically a relatively long portion of a stream, *e.g.*, one minute. The longer are the chunks, the lesser is the overhead of control messages.

A client is connected to a set of peers, which are expected to be able to serve it. Hence these peers belong to the sector corresponding to the chunk that the client is willing to download now. We note by $\Upsilon(z)$ the set of peers to which a client z sends requests. From a cycle t to a cycle $t+1$, the set $\Upsilon(z)$ is totally refreshed, because the chunk requested at cycle t is different from the chunk requested at cycle $t+1$. When a client wants to download any past portion of the stream, it should first determine the sector associated with the first chunk of this portion. After it finds a peer that has stored the requested chunks in this sector, it should then jump to the next sector in order to retrieve the next chunk and continues consuming the stream.

Besides, a peer contacted by a client is expected to be able not only to deliver the requested chunk, but also to determine a set of peers this client can contact in order to fetch the next chunk. Peers regularly exchange information for that purpose. The overall overlay formed by the turntable structure depends on how peers are linked inside a sector, which are called the *intra-sector links*, and how peers from one sector are linked to peers of the next sector, namely the *inter-sector links*. We denote by $\Gamma^{intra}(x)$ (resp. $\Gamma^{inter}(x)$) the set of intra-sector neighbors (resp. inter-sector neighbors) of a peer x.

The source must be connected to one peer in every sector. These peers are called *representative*. We denote by $\hat{x}_i$ the representative for sector s_i. When it is time for a sector s_i to handle a new chunk, the source alerts the representative $\hat{x}_i$ and sends the chunk to it, then this chunk is diffused in the sector. During the m next cycles, this chunk is called the *fresh chunk* for this sector.

Several algorithms runs the turntable. First, an algorithm for the *diffusion of fresh chunks*. As the fresh chunks are also the most requested chunks, they should be diffused as quickly as possible to many peers within the sector. Second, an algorithm ensuring *a fair repartition of the past chunks*. Ideally, the number of replicas of a chunk should correspond to the number of requests emitted for this chunk. Third, an algorithm for *finding a peer storing a requested chunk*. When a peer is contacted by a client for a chunk, either this peer is able to serve the client (it has the chunk and it has the capacity to upload the chunk), or it should find a peer that is able to serve the client. We detail these algorithms in Sec. II-C.

B. Inter- and Intra-Link Management

We present now a set of distributed algorithms for neighborhood management in the turntable overlay.

Intra-Link Management. We choose to build a lightweight overlay within every sector. We use a gossip-based technique inspired by T-Man [8]. Every peer is connected to a subset of peers, which it continuously refreshes. Peers periodically exchange messages, which carry neighborhood information. Then, every peer connects to the "best" peers among its current neighbors and all the possible neighbors described in these messages. In the resulting overlay, every peer is connected to the peers that it considers as the best.

We propose that peers connect preferentially to the neighbors that store collectively the largest set of distinct chunks. Indeed, the larger is the set of distinct chunks at one hop, the higher is the probability to find a requested chunk at one hop. In other words, we give priority to the performances of the algorithm that aims to find a requested chunk. For combinations of peers having almost similar performances, peers discriminate by selecting the closest peers in the network. Here the motivation comes from ISP-friendly considerations.

Inter-Link Management. The inter-sector links aim to ease the retrieval of consecutive chunks. A client z retrieving a past stream portion requests consecutive past chunks. The purpose

[1]in the following, we omit the modulo for notation clarity.

of an inter-sector link is to connect two peers that are in consecutive sectors and that store some successive chunks from past stream portions. When z is served by a node $x_i \in s_i$, its next request is in sector s_{i+1}, *i.e.* z should find in sector s_{i+1} a peer that stores the chunk next to the one it just downloads from x_i. Ideally, the next chunk is stored by a peer x_{i+1}, which is in $\Gamma^{inter}(x_i)$. Thus x_i can introduce x_{i+1} to z and save request messages. More generally, the set $\Gamma^{inter}(x)$ is constructed with a gossip-based mechanism, where the number of chunks stored by peers that are next to chunks stored by x allows to choose the neighbors in the next sector.

C. Algorithms

We describe now the algorithms that are implemented on top of the turntable overlay. Please note that various protocols can be designed. We present here the ones that have demonstrate good performances during our simulations.

1) Finding a peer storing a chunk: A client z is connected to a set of peers $\Upsilon(z)$ in the sector that is responsible of the requested chunk. This set of peers has been given to z by peers from the previous sector. If none of these peers is able to serve z (either because neither they have the requested chunk, nor they have the capacity to upload it), a search should be done in the whole sector. We use here a classic two-steps algorithm where, first, the intra-sector neighbors of every peer in $\Upsilon(z)$ are explored, and, if the chunk is still impossible to download, then the sector overlay is flooded at k hops.

2) Fresh chunk management: We use a *push-based* approach to distribute fresh chunks. Our approach is to distribute chunks through a gossip process based on two parameters. Peers receiving a chunk ch_j decide to forward it with a probability p_j, if this chunk has not been already forwarded more than a given number of times. The computation of p_j is an issue. Our approach consists in leveraging the knowledge of the popularity of fresh chunks in the previous sector to adjust p_j. We denote by f_j the failure frequency of a fresh chunk ch_j, *i.e.* the number of requests for ch_j that have not been fulfilled. The idea is that f_j is probably similar to f_{j-1} if the probability p_j is close to p_{j-1}. Therefore, if the failure ratio f_{j-1} is too high, the probability p_j should increase, and *vice versa*. We propose then to use an *Additive Increase Additive Decrease* (AIAD) mechanism to adjust the number of fresh chunk replicas. When a peer x gets the knowledge of a ratio of request failures lower than a given threshold f_{low}, it can decide to decrease the probability by a value a. When x infers that the ratio of request failures is higher than a given threshold f_{high}, it can increase the probability by another value b. Otherwise, x maintain the forwarding probability unchanged. The goal of the control algorithm is to keep the number of request failure within a prefixed quality of service.

3) Past chunk management: The limitation of the storage capacities imposes to not create a replica of each chunk at each peer. An important issue for past chunk management is the choice of the chunk to be removed when the local storage is full and when a new chunk should be stored. We implement a *pseudo-LRU* algorithm where a peer first establishes a list of chunks that it can discard because it knows that at least one replica exists in the sector, then, it determines the chunk to remove among this selected chunks by a classic *least recently used* (LRU) policy. Now, we study the creation of new replicas. Our proposal is inspired by [9] where authors describe a distributed replication algorithm with regards to the popularity of data item and storage capacity of peers, as well as the heterogeneity and dynamics of network and workload. In the same idea that for fresh chunks, every peer x maintains a counter of the number of failed received requests f_{xi} for a chunk ch_i. When a failure ratio f_{xi} is above a given threshold, the peer x decides that the chunk ch_i is not replicated enough. Then, the chunk ch_i is put in the *set of missed chunks* of x. Periodically, the peer x picks one chunk in this set, and requests it in its sector, so that a new replica can be created.

III. Simulations

We implement our turntable time-shifted system on PeerSim, a simulator targeting large-scale and dynamic overlays.

Two sets of studies conducted in 2008 and 2009 have been utilized to model the behavior of *shifters* (viewers of catch-up TV). The first set is real measurements by DVR vendors, which have been given in a Nielsen report [10]. The second set of related works is the recent measurements conducted on IPTV [11] and VoD systems [12].

The TV prime-time is clearly on evening. Measurements made in [10] confirm that shifters are obviously more connected at certain time of the day than others. In our simulator, we create x new peers at every cycle. During less attractive hours, x equals 1, while it can be equal to 10 at the prime-time.

With respect to the measurements in [11], peers get assigned a role in our simulation: half of the peers are *surfers* (watch a same program during 1 or 2 chunks before to switch to another program), 40% of them are *viewers* (switch after a duration uniformly chosen between 2 and 60 minutes), and only 10% are *leavers* (stay on a program during a time comprised between 60 and 1000 chunks, *i.e.* a TV constantly opens during up to 20 hours).

Several continuous chunks form a program, which is associated with a *genre* [11], a popularity chosen in a predefined distribution, and a length ranging from 30 to 100 chunks. Three genres are considered: 80% are *free*, 15% are *news* and 5% are *kids*. While a program reaches its predefined length, a new program is immediately created. As it has been noticed in various IPTV measurements, a viewer is more likely to choose a program in the same genre when it switches.

Based on the configurations of user behavior and program popularity, Figure 2 represents the Cumulative distribution Function (CDF) of the lag of shifters at the end of our simulation. The embedded plot zooms on the $5\,000$ first minutes, which represents actually more than 80% of shifters. A point at $(1\,000, 0.50)$ means that half of shifters are watching a program broadcasted less than $1\,000$ minutes ago. Note on the embedded figure that variable program popularity results in a sinuous curve. This curve is actually conform with the recent measurements made in [10].

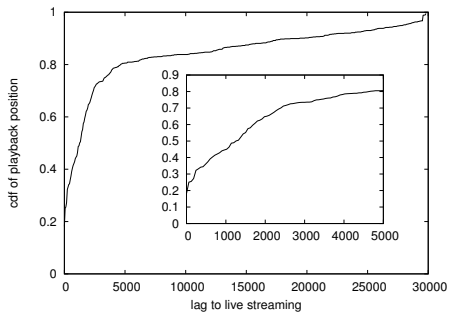

Fig. 2. CDF of playing positions

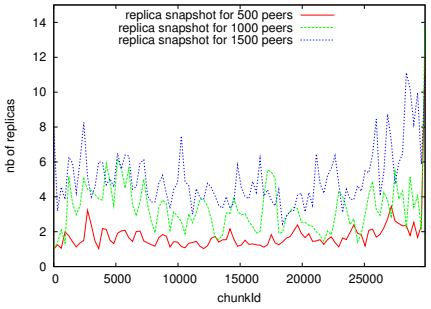

Fig. 3. Average number of replicas per chunk during the simulation

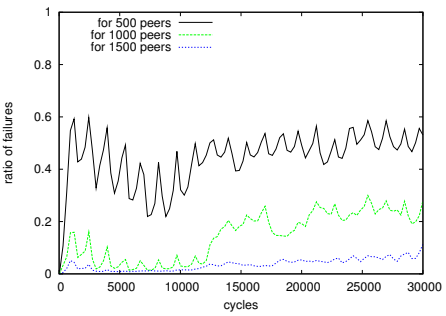

Fig. 4. Ratio of failures per demand

Our simulation runs 30 000 cycles, *i.e.*, more than 20 days. We assume that the turntable has $m = 20$ sectors, each of which introduces one representative. Each peer, randomly assigned to a sector, can store 100 chunks. A peer has privileged relationships with 5 intra-sector neighbors, and 5 inter-sector neighbors that are both chosen among 20 acquaintances. We consider that a chunk represents one minute of stream in our simulator.

In general, the upload capacity of peers is hard to set because they are highly dependent on the service. In our system, peers are expected to upload a whole chunk to other peers, *i.e.*, one data transmission consists of one-minute long stream delivery. Therefore, the capacity of a peer is described as the number of concurrent streams that the peer is able to send to other peers. Obviously, if the stream source generates High-Definition TV (HDTV) content, and if peers are end users' computer, the average capacity can not be large. Here, we set the average capacity of peers as 1 (in average, a peer is able to send one stream to only one peers).

A. Simulation Results

1) Results for Chunk Replication: In Figure 3, we compute the average number of replicas for every chunk during all the time this chunk has been in the system. Then, we compare three cases with a variable number of peers, from 500 to $1,500$. The variation of the number of replicas is quite high, mostly because the popularity of chunks varies a lot from a chunk at the beginning of a popular program to a chunk at the end of a unpopular program. However, every chunk has at least one replica in the system, therefore, the availability of chunks is guaranteed. As expected the number of replicas is higher when the number of peers increases.

2) Quality of Services - Fulfilled Requests: In this last part, we observe the quality of service. We distinguish a *flooding*, when all peers of a client can not treat the request of the client and issues a k-hop flooding, and a *failure*, when even the flooding fails, *i.e.* the client can not be served. In the former case, the overhead generated by the request messages is important. In the latter case, the system is unable to serve the client. Figure 4 shows the evolution of the ratio of failures to the number of received requests during the simulation.

The number of peers has a dramatic impact on the quality of services. When n is equal to $1,500$ (approximately the maximum number of concurrent clients), the number of failures is still low (less than 5% of requests are not fulfilled). Actually, most failures occur for fresh chunks, where, despite the algorithm for the diffusion of fresh chunks, peers have not the capacity to generate the number of replicas on time. For smaller number of peers, the problem of congestion becomes more important, With an average upload capacity of 1, peers can only serve as many clients as n. We observe that the ratio of failed requests is approximately the ratio of the number of clients to the number of peers. The system is able to almost entirely utilize the upload resources of peers.

REFERENCES

[1] X. Yang, M. Gjoka, P. Chhabra, A. Markopoulou, and P. Rodriguez, "Kangaroo: Video Seeking in P2P Systems," in *Proc. of IPTPS*, 2009.
[2] X. Wang, C. Zheng, Z. Zhang, H. Lu, and X. Xue, "The design of video segmentation-aided VCR support for P2P VoD systems," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, May 2008.
[3] J. Zhuo, J. Li, G. Wu, and S. Xu, "Efficient cache placement scheme for clustered time-shifted TV servers," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1947–1955, November 2008.
[4] W. Xiang, G. Wu, Q. Ling, and L. Wang, "Piecewise Patching for Time-shifted TV Over HFC Networks," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 3, pp. 891–897, Aug. 2007.
[5] C. Huang, C. Zhu, Y. Li, and D. Ye, "Dedicated Disk I/O Strategies for IPTV Live Streaming Servers Supporting Timeshift Functions," in *Proc. of IEEE CIT*, 2007, pp. 333–338.
[6] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller, "LiveShift: Peer-to-Peer Live Streaming with Distributed Time-Shifting," in *Proc. of 8th Int. P2P Conf.*, 2008, pp. 187–188.
[7] D. Gallo, C. Miers, V. Coroama, T. Carvalho, V. Souza, and P. Karlsson, "A Multimedia Delivery Architecture for IPTV with P2P-Based Time-Shift Support," in *Proc. of 6th IEEE CCNC*, 2009, pp. 1–2.
[8] M. Jelasity and O. Babaoglu, "T-man: Gossip-based overlay topology management," in *ESOA, Intl'l Work. on Engineering Self-Organising Systems*, 2005.
[9] M. Sozio, T. Neumann, and G. Weikum, "Near-optimal dynamic replication in unstructured peer-to-peer networks," in *Proc. of the 27th ACM Symp on Principles of Database Sys. (PODS)*, 2008.
[10] Nielsen, "How DVRs Are Changing the Television Landscape," Nielsen Company, Tech. Rep., April 2009.
[11] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatrianin, "Watching television over an ip network," in *Proc. of Usenix/ACM SIGCOMM Internet Measurement Conference (IMC)*, 2008.
[12] Y. Hongliang, Z. Dongdong, Z. B. Y., and Z. Weimin, "Understanding user behavior in large-scale video-on-demand systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, pp. 333–344, 2006.